TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Experimentální backend systém pro dobíjecí stanice

Diplomová práce

Studijní program: N2612 Elektrotechnika a informatika
Studijní obor: Informační technologie

Autor práce: **Bc. Jaroslav Hrabal**
Vedoucí práce: Ing. Pavel Jandura, Ph.D.
Ústav mechatroniky a technické informatiky

Konzultant práce: Ing. Ivan Menoušek
Siemens s.r.o.

Liberec 2022

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání diplomové práce

Experimentální backend systém pro dobíjecí stanice

Jméno a příjmení: **Bc. Jaroslav Hrabal**
Osobní číslo: M18000140
Studijní program: N2612 Elektrotechnika a informatika
Studijní obor: Informační technologie
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2021/2022**

Zásady pro vypracování:

1. Pokračujte v rešerši existujících řešení systému centrální správy pro dobíjecí stanice a problematiky zavádění nových revizí protokolu OCPP.
2. Rozšiřte vaši stávající implementaci protokolu a komunikaci zabezpečte proti útokům zvenčí.
3. Navrhňete frontend zobrazení, které bude komunikovat s uživatelem.
4. Navržené řešení otestujte s reálnou dobíjecí stanicí.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	40--50 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Schmutzler Jens Andersen Claus Amtrup Wietfeld Christian Distributed energy resource management for electric vehicles using IEC 61850 and ISO/IEC 15118. In Proceedings of 8th IEEE vehicle power and propulsion conference (VPPC) Seoul, SouthKorea, 201 2s. 9 12 DOI: 10.1109/EVS.2013.6914751

[2] Open Charge Alliance online Arnhem, Nizozemsko, Business park Arnhem Buiten © 2018 [cit. 20.7.2020]. dostupné z: https://ww w.openchargealliance.org/protocols/ocpp 15

[3] Juan Rubio, Cristina Alcaraz, Javier Lopez . Addressing security in OCPP: Protection against Security in OCPP: Protection against man in the middle attacks. In: 2018 9th IFIP International Conference on New Technologies, Mobility and Security NTMS Paris, France

Vedoucí práce:	Ing. Pavel Jandura, Ph.D. Ústav mechatroniky a technické informatiky
Konzultant práce:	Ing. Ivan Menoušek Siemens s.r.o.
Datum zadání práce:	12. října 2021
Předpokládaný termín odevzdání:	16. května 2022

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Černohorský, Ph.D.
vedoucí ústavu

V Liberci dne 12. října 2021

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

Bc. Jaroslav Hrabal

Poděkování:

Na tomto místě bych rád poděkoval všem, s jejichž pomocí tato práce vznikla. Zvláště chci poděkovat vedoucímu práce Ing. Pavu Jandurovi, Ph.D. Dále bych rád poděkoval také Ing. Ivanu Menouškovi a rovněž své rodině za poskytnutou podporu.

Abstract

Diplomová práce se zabývá objektově orientovaným programováním. Zaměřuje se na jeho užívání a na návrh serveru pro dobíjecí stanice. Pozornost bude věnována backendu, ale také frontendu a databázovému uspořádání. Program byl realizován s použitím jazyka Java a frameworku Spring. Pro jeho sestavení byl použit Maven a navrhování proběhlo ve vývojovém prostředí Intelij IDEA.

V první části práce jsou teoretické poznámky, které je potřeba vědět pro vyhotovení práce. Je zde popis OCPP (Open Charge Point protocol), popis Javy a jeho frameworku Spring, a také návrhové vzory, podle kterých lze programovat. Druhá část diplomové práce se zabývá jeho reálnou implementací. Jsou zde řešeny návrhy vzorů, implementované protokoly a knihovny, postup řešení a uvedení do provozu. Postupně se navrhuje od postupu projektu do jednotlivých funkcí a databázových modelů. Následuje testování a uvedení do provozu. Na závěr budou zhodnoceny úspěchy a neúspěchy práce.

Abstract

This thesis is using object oriented programming. Its main focus it to use this programming strategy f or a design of a server for charge stations. It will be solving the problem of implementing a backend server, as well as its usage and database structure. Program was implemented through the use of Java and its framework Spring. Maven was used for the build. IntelliJ IDEA was the developing environment.

The first part of the thesis contains theoretical notes, which are necessary to know for the completion of the work. There is a description of OCPP (Open Chage Point Protocol), description of Java and its framework Spring, and design pattern that teach of proper practices in work and design of models. Secon part is focusing on real implementation. It has the actual solutions and designs to the real problem. As well as how to test it and implement it.

Obsah

Seznam obrázků, grafů a tabulek:

Seznam symbolů, zkratek a termínů

OCPP	Open Charge Point Protocol
OCA	Open Charge Aliance
JVM	Java Virtual Machine
JRE	Java Runtime Environment
MVC	Model View Controller
CRUD	Create Read Upgrade Destroy
SOAP	Simple Object Access Protocol
JSON	JavaSript Object Notation
HTML	Hypertext Markup Languauge
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface

1. Úvod

Tato práce se zabývá problematikou vývoje infrastruktury pro elektromobilitu. Jedná se o vývoj serveru, který bude komunikovat s dobíjecí stanicí, bude ji řídit a uchovávat její informace pro pozdější zpracování. Cílem práce bude vytvořit server a jeho ovládání pro dobíjecí stanici na prostorách university.

1.1 Motivace

Cena nafty a benzínu stále stoupá. Dopravu používáme všichni a je v našem nejlepším zájmu ji udělat tak dobrou, jak to jde. Proto je potřeba se podívat po nových možnostech, jak zajistit dopravu. Elektromobily jsou již na světě, ale potřebují lepší zázemí. Stanic, kde lze načepovat energii, není mnoho. K vybudování infrastruktury je potřeba mnoho věcí a jednou z nich jsou právě servery, na kterých budou stanice operovat.

1.2 Náplň práce

V průběhu práce bude rozebrána problematika, která je potřebná k naprogramování takového serveru. Bude se diskutovat jak o teorii, tak o praktickém programování. Začne se nejdříve s teorií, kde budou veškeré informace potřebné pro druhou, praktickou část.

1.2.1 Teoretická část

Nejprve bude představena teorie potřebná k sestavení serveru. Proberou se protokoly a standardy, které je potřeba při vývoji dodržet. Dále bude popsána technologie použitá v průběhu práce, proběhne představení programovacích jazyků, a vývojových prostředí, která je mohou implementovat.

Kromě teorie o samotných nástrojích budou rovněž uvedeny návrhové postupy, podle kterých se programuje. Jedná se jak o vzory, které se dodržují jako programovací standard pro vývoj aplikací, tak o postupy, kterých se držíme, abychom se k tomuto výsledku dostali.

Po teorii vývoje bude následovat teorie bezpečnosti. Zde bude pojednáno o tom, jaké mohou být hrozby a jak jim zabránit. Bude se jednat jak o obecné informace, tak i o konkrétní bezpečnost zvoleného frameworku.

Téma bezpečnosti pak uzavře teoretickou část práce a bude následovat praktická část. Bude se pokračovat podle teorie a jejich nejlepších metodik.

1.2.2 Praktická část

V praktické části se začne návrhem. Zvolí se jeden z předvedených vzorů. Práce pak bude následovat ve vývoji tohoto vzoru.

Bude se jednat o požadavky, které práce má, a co by takový program měl mít. Postupně bude rozkryto, co je potřeba implementovat a za jakým účelem. Rozebere se protokol, který definuje, jak by program měl vypadat, aby byl kompatibilní s ostatními, a v práci pak tyto pokyny budou dodržovány.

Dále budou uvedeny nutné věci, které jsou potřeba pro programování a tvorbu práce. Bude potřeba jak hardware, tak software. Dále budou zvoleny jazyky a vývojová prostředí pro vývoj.

Následuje vlastní návrh práce, struktura programu, a nastínění toho, jak by měla práce vypadat. Budou dodrženy návrhové vzory a postupy, které jsou potřebné pro přípravu programování. Rovněž bude navržena forma celé aplikace.

Následně dojde na vlastní programování, vytvoření projektu, souborů, balíčků a provázání knihoven. Přijde k psaní souborů a funkcí, a vývoj aplikace.

Po vývoji bude zmíněno testování, spouštění aplikace a její používání. Bude zde uvedeno, jak se s programem zachází a jak ho udržovat.

V závěru práce se nachází shrnutí toho, čeho se podařilo nebo nepodařilo dosáhnout.

2. Použité technologie

V této kapitole se budu zabývat technologiemi, které byly použity. Budou zde uvedeny veškeré technologie, jejich výhody a důležité vlastnosti. Jejich popis nebude ale příliš detailní, jelikož jsou přístupné v podobě mnoha zdrojů na internetu.

2.1 OCPP

OCPP (Open Charge Point Protocol) byl vytvořen za účelem kompatibility pro elektrické vozy a jejich zázemí. Protokol byl aktivní už od roku 2009. Od té doby se OCPP stal jediným protokolem pro elektromobilitu. Bylo řečeno [1], že *OCPP* je důležitý standard, který umožňuje jednotnou komunikaci mezi dobíjecími zařízeními. To je prospěšné nejen pro uživatele, ale také pro provozovatele. Když mají již systém a chtějí si objednat nový, *OCPP* zajistí, že budou kompatibilní, pokud se bude jednat o stejnou verzi.

2.1.1 Protokol verze 1.5

Tento protokol dostal první reálné využití jako nový výtvor firmy E-Laad v Nizozemí. A byl nejdříve vytvořen jako fórum pro diskuze. Jeho účelem bylo vytvořit otevřený standard pro komunikaci, aby jejich dobíjecí systémy mohly fungovat na jednom protokolu. Open Charge Alliance, která byla vytvořena E-Laad, se pořád stará o toto fórum, které vytvořilo *OCPP* protokol. Je to velice důležitá metodika nutná k optimalizování cen a rizik.

2.1.2 Protokol verze 1.6

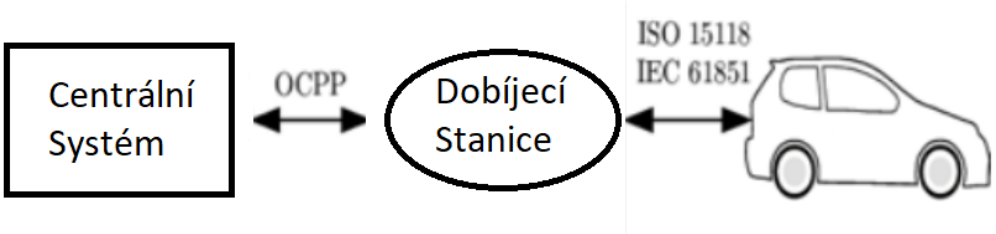
OCA (Open Charge Alliance) vyhlásila v roce 2015 vývoj nové verze protokolu 1.6. Nejdříve bylo doporučeno používat starý protokol 1.5, který byl stále zdokonalován i při vývoji protokolu 1.6. Vývoj verze 1.6 probíhal od května do října, kdy byl oficiálně vydán a splnil většinu právě potřebných dovedností.

2.1.3 Protokol veze 2.0.1

OCPP 2.0 byl odstartován o něco později. Nebyl volně k použití až do roku 2018. Mnoho skupin pomohlo ve vývoji této verze sdílením svých vylepšení a zkušeností. Spolu s těmito vývojáři se podařilo vylepšit protokol na verzi 2.0.1., která pomohla odstranit nedostatky v bezpečnostním sektoru.

2.1.4 OCPP porovnání

OCPP je široce využívaný protokol pro dobíjecí stanice a jejich komunikaci s centrálním systémem. K jeho rozšíření nejvíce přispěla OCA. Jedná se o organizaci, která podpořila rozvoj inovací a spolupráci v rámci problematiky elektromobility. Protokol není závislý na prodejci, proto si mohou vlastníci stanic a podnikatelé zvolit své vlastní prodejce. *OCPP* zaručí, že systémy budou funkční od jakéhokoliv výdejce.



Obrázek 1: Schéma zapojení dobíjecí stanice

Protokol byl implementován v roce 2010. Od této doby už ho začala používat řada prodejců dobíjejících stanic. Pro rok 2022 jsou používané 3 hlavní verze:

- z roku 2012 verze 1.5 podporující pouze *SOAP* (Simple Object Access Protocol)
- z roku 2015 verze 1.6 podporující *SOAP* a *JSON (*JavaSript Object Notation)
- z roku 2018 verze 2.0 podporující pouze *JSON*

V této práci se bude používat verze 1.5, protože je implementována v dobíjecí stanici poskytnuté pro její zhotovení.

2.2 HTTP

Tento protokol byl vyvinut pro použití na *WWW* (World Wide Web). *HTTP* (Hypertext Transfer Protocl) je protokol, který umožňuje posílat různé zdroje, jako jsou například dokumenty *HTML* (Hypertext Markup Langage). Většina internetu je založena na této technologii posílání žádostí a odpovědí. *HTTP* je bezstavový, tím mohou být žádosti unikátní. Výhodou je, že server nemusí ukládat informace o každém klientovi. Na druhou stranu je nutné přidat stejné informace do každé žádosti.

2.3 WebSocket

WebSocket umožňuje okamžitou komunikaci mezi klienty a servery. Umožňuje poslat zprávy v jakýkoliv okamžik. Díky *WebSocket* můžeme snížit prodlevu při zpracování dat. Také můžeme připojení kdykoliv vypnout a zapnout.

Připojení *WebSocket* začíná žádostí klienta pro server. Tato žádost by měla obsahovat hlavičku *Connection: Upgrade*. Pokud server tuto žádost potvrdí, hlavička *Sec-WebSocket-Accept* je vrácena společně s *Sec-WebSocket-Key*.

Klient a server se musí dohodnout, jaké podprotokoly budou používat při výměně prvních žádostí. Server by měl zvolit vyhovující podprotokoly a vrátit je klientovi. Mezi tyto výměny patří hlavička *Sec-Web-Socket-Protocolu*, který je používán pro žádost *OCPP 1.5*.

2.4 Java

2.4.1 Počátek Javy

Java byla vytvořena firmou Sun Microsystems. V roce 1990 se kvůli nespokojenosti programátorů tato firma rozhodla vytvořit jazyk, který by byl dostačující pro její zaměstnance. Systém, který firma používala, měl spoustu nedostatků, zejména *NeWS* software, který byl právě v této době vytvářen, neměl velký ohlas.

Proto byl povolán do práce tým šesti programátorů. Pod názvem Green se tato skupina zabývala problematikou propojení a interakcí mezi nástroji firmy. Brzo se zjistilo, že nástroje, se kterými pracovali, například laserové přehrávače disků a stereo systémy, pracovaly na odlišných procesorech. Z tohoto důvodu musel být zvolen naprosto nový přístup k programování. Tým Green následně začal vyvíjet nový objektově orientovaný programovací jazyk, který byl nazván *Oak*. Ten byl potom v roce 1995 ohlášen jako nový programovací jazyk *Java*.

2.4.2 Vlastnost Javy

První důležitou vlastností je, že programovací jazyk Java je založen na třídách a je objektově orientovaný. Díky tomu umožňuje seskupování objektů, což může být použito i ve více projektech. To vede k šetření času při programování.

Dále je *Java* souběžný jazyk. Je možné spustit více programů nebo částí programů naráz. Tím dochází ke zlepšení efektivity provozu.

Třetí na řadě je přenositelnost *Javy. Java* funguje na ideologii „napiš jednou, použij všude“. To znamená, že jakmile se napíše kus kódu, tak se může použít na ostatních projektech. Program se nemusí psát od začátku znovu.

Poslední důležitou vlastností je bezpečnost *Javy.* Veškerý kód je převeden do bytekódu, který není nemožné přečíst. Tím chrání kód před nedůvěryhodnými zdroji a viry.

2.4.3 Výhody Javy

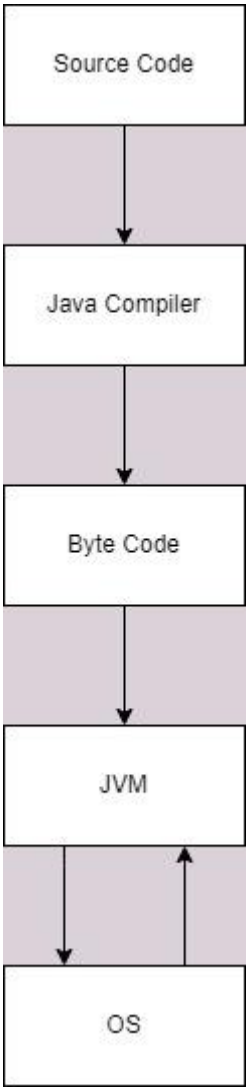
Java je objektově orientovaný jazyk a zároveň je jednoduchým prostředkem. Lze se ji naučit snadno a rychle. Na rozdíl od jiných jazyků *Java* používá vestavěný *interpreter*. Může být použita na více platformách a operačních systémech. Dále má vlastní manipulaci paměti a údržbu objektů. Uživatelé jsou tak osvobozeni od potřeby manipulovat s explicitní pamětí manuálně. Je možné dosáhnout vysokého výkonu správným použitím vícevláknových procesů. Aplikace jsou také jednoduché na údržbu a změny, moduly mohou být snadno staženy z internetu se svými vlastními bezpečnostními systémy zabraňujícími nepovolenému přístupu a virům.

2.4.4 Java Development Kit

Jedná se o plnou sestavu nástrojů potřebných pro vývoj Java aplikací. Obsahuje *Java Runtime Environment, compiler debuggery* a dokumenty s tím spojené. Pro vývoj *Java* aplikací je potřebné mít nainstalovanou *Java Development Kit*.

Aby bylo možné používat nástroje pro vývoj, je také potřeba mít prostředí, ve kterém se to bude spouštět. K tom slouží *Java Runtime Environment*. Tato verze poskytuje minimum potřebné ke spouštění Java aplikací. A už je součástí *Java Development Kit* při instalaci.

2.4.5 Java Virtual Machine



Obrázek 2: JVM

Tato část je nezbytná pro tvorbu a spouštění *Java* aplikací.

Je složená ze dvou fází:

- kompilace kódu – *Java Development Kit* už přichází s *JAVAC* kompilátorem. Ten přeloží kód a vytvoří speciální formát *Java Runtime Environment* může použít. Takovýto překlad kódu nazýváme bytecode.

- spuštění kódu – Java Virtual Machine spustí bytecode, který kompilátor vytvořil. Ten je spustitelný na jakékoliv platformě. Tím je lehce přenositelný.

2.4.6 Garbage collector

V Javě se nemůžou mazat objekty, které se vytvořily, nebo paměť která už se alokovala. K tomu slouží *Garbage Collector*. Ten je součástí Java Virtual Machine. O paměť se stará sám a automaticky maže objekty, které už nemají odkazy. Proto by si měl programátor dát pozor na to, jaké objekty používá, jestli nepoužívá objekty, které byly v provozu po delší dobu. Není možné tento proces ovlivnit.

2.4.7 Classpath

Zde se nachází veškeré soubory, které Java Virtual Machine potřebuje pro spouštění a kompilaci. Je nutné cestu definovat dobře, jinak nebude vědět, odkud programy spouštět.

2.4.8 Použití Javy

Program *Javy* je rozdělen do souborů obsahujících třídy. Třídy mohou potom reprezentovat objekty, se kterými manipulujeme. Tomu se říká objektově orientované programování.

K tomu, aby třída mohla vytvořit objekt, musí mít *constructor*, který umožní vytvoření instance třídy. Tento objekt pak reprezentuje naprogramovanou věc. Pro získání dat z objektů se používá metoda typu *getter* a pro změnu dat slouží *setter*.

Java má několik vlastností, na které se musí držet dohled:

- *děditelnost* – objekt může zdědit vlastnosti svého rodiče. Tato skutečnost dělá kód použitelným z jednoho místa.
- *polymorfismus* – je možné předělat objekt, aby měl více stejných metod. Tento princip se pak dělí do dvou kategorií:
    - overloading – nastane v případě, kdy má několik metod ve stejné třídě stejné jméno;
    - overriding – pro tento typ musejí být parametry metod stejné, jak pro zděděnou třídu, tak pro rodiče.
- *abstrakce* – tato podmínka se splní, pokud definujeme pouze funkcionalitu třídy, ale nevyplníme konkrétní informace o ní. Může pak dojít k dělání typů tříd a toho využívají následující frameworky.

- *zapouzdření* – pokud necháme data pouze v jedné třídě, můžeme ji takto zapouzdřit. Java *bean* je plně zapouzdřený, více informací se nachází v další kapitole.

2.4.9 Spring Framework

Jedná se o *open source* framework *Javy* poskytující pomoc při vývoji aplikací. Jedná se o jednu z nejvíce populárních možností. Programy psané v jazyku *Java* jsou obvykle těžkopádné. Mají spoustu komponentů závisejících na operačním systému, které mohou programování zpomalovat. Spring je považován za levný, flexibilní a bezpečný. Efektivně využívá systémových zdrojů. To snižuje práci při vývoji aplikací.

Spring nabízí možnost uspořádat kód do tří kategorií. Konkrétně je to prezenční, datová a logická vrstva. Bez frameworku je obvykle kód sestaven v kupě. To není nejlepší praxe. Spring pomáhá uspořádat kód tak, aby byl přehledný a jednoduchý k použití. Jeho hlavní návrhový vzor je *depenency injection*, návrhový vzor, který umožňuje programátorům psát více rozvrstvenou architekturu. Znamená to, že Spring rozumí různým *Java anotacím* vývojář uvede nad objekty a třídy a Spring je potom uspořádá a vytvoří sám. Toho se docílí napsáním následujících anotací:

- @component – anotace slouží k označení objektů a tříd, které by měl Spring spravovat;
- @autowired – spring započne hledání ostatních anotací a jednotlivých komponent.

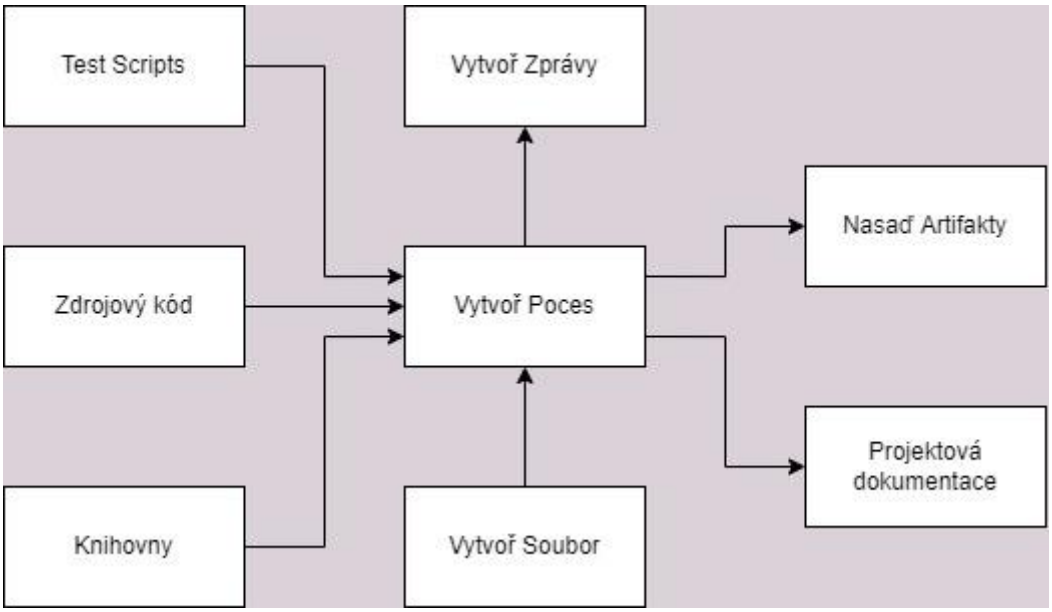
2.4.10 Spring Boot

Používá se k vytváření mikro servisu. Mikro servis je struktura umožňující vývojářům vyvinout a nasadit nezávislé servisy. Každý běžící servis má svůj vlastní proces, tím docílí jednoduché implementace aplikací. Mikro servisy nabízejí různé výhody. Mezi tyto výhody patří jednoduché nasazení, snadná škálovatelnost, minimální nutnost konfigurace, kompatibilita s kontejnery a rychlejší vývoj.

Spring Boot sám nastaví aplikaci podle toho, jaké má projekt aktivní vazby. Pomocí anotace @EnableAutoConfiguation dojde ke konfiguraci projektu. Následně se uvede do provozu napsáním anotace @SpringBootApplication nad hlavní metodou spouštějící aplikaci. Dále je důležitá anotace @ComponentScan, která vyhledá všechny komponenty v projektu.

2.4.11 Maven

Maven je nástroj pro řízení projektu. Je napsán v *Javě*, ale dá se použít i pro projekty Ruby, C#, Scala a další. Umožňuje vývojářům vytvářet projekty, závislosti a dokumentaci. Maven dodržuje následující proces tvorby:



Obrázek 3: Maven

2.5 Návrhové vzory

Kromě programovacích jazyků je také nutnost představit návrhové vzory, podle kterých se aplikace bude vyvíjet. Zde se nacházejí postupy programování pro přístup k celkovému návrhu. Dále také programové návrhy, výhody a nevýhody.

2.5.1 Model Vodopád

První návrhový model spočívá v lineárním postupu navrhování. Problém se rozdělí do několika po sobě jdoucích etap. Ty se následně zhotoví jedna za druhou. Tento model bývá jedním z méně flexibilních a opakovatelných vzorů. Typický způsob je rozdělit návrh do etap početí, zahájení, analýza, design, tvorba, testování, nasazení, údržba.

Jednotlivé fáze objektu se dále mohou definovat přesněji. První část může obsahovat požadavky na software. Z analýzy se dají vytvořit modely, schémata a principy podnikání. Samotný návrh pak ukazuje architekturu softwaru. Vlastní tvorba se dělí na vývoj, dokazování a integraci software. Nakonec se jedná o instalaci, migraci, podporu a údržbu hotového systému.

Pohybujeme se pak z jednoho stádia vývoje do dalšího pouze pokud jsme dokončili předchozí nutnosti. Investováním času do studie požadavků a návrhů se může předejít ztracenému času, kdy se chyba najde příliš pozdě.

2.5.2 Model Spirála

Zatímco Vodopád je preferovaným návrhovým modelem pro zákazníky a menší projekty, Model Spirála je používán zejména většími firmami pro dlouhodobé plány. Jeho princip spočívá ve čtyřech hlavních etapách.

Prvním je upřesnění cílů a identifikace alternativních řešení. Druhým pokynem je identifikace a odstranění risků. Blížíme se potom do třetího sektoru, kde vytvoříme novou verzi modelu. Následně se ukončí spirála poslední částí pro revize a plány pro další kolo.

Nevýhodou tohoto modelu je jeho cena. Nevyplatí se dělat u menších projektů. Dále je model velmi závislý na odhalení risků. Bez expertů v oboru by mohl být takovýto postup špatná volba.

2.5.3 Model Přírůstku

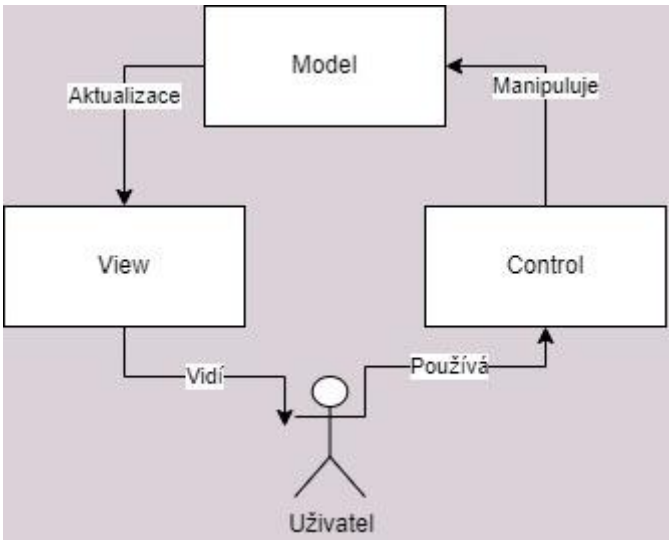
Poslední model spočívá v tom, že se vytvoří minimální verze programu jen s několika možnostmi a opakovaným postupem se program vyvíjí do větší škály. Model nemusí pokračovat konstantně. Je možné vyvinout novou verzi jen když je potřeba.

Nejdříve se začne s nezbytnými operacemi, které by program měl být. A postupně přidáváme více možností. Po vytvoření nové verze je program dán zákazníkovi. Potom se může diskutovat o programu a o tom, zda je nová verze nutná.

Tímto postupem se může vytvořit software rychle a je přístupný co nejdříve. Zákazník ví, co se děje v projektu. Změny se dají aplikovat rychle. Nevýhodou je nutnost mít zkušený tým vývojářů. Cena vývoje je také vysoká, a to kvůli repeticím. Více o možnostech popisuje Systems Development Life Cycle (SDLC) [2].

2.5.4 Model View Controller

Aplikace v tomto návrhovém vzoru by měla být rozdělena do tří hlavních částí. Jedná se o datový model, princip kontroly a informace, které zobrazujeme. Všechny tyto tři části by měly být odděleny do jednotlivých objektů. Jedná se jen o část, kde uživatel interaguje s programem, model nezahrnuje další logiku aplikace.



Obrázek 4: MVC

View je jen pro zobrazení dat. Neměla by zde být žádná logika kromě toho, jak zpřístupnit data. Model by měl obsahovat pouze aplikační data, ne způsob, jak je zobrazit. Poslední část je pro logiku programu. *Controller* naslouchá tomu, co přijde od zobrazení a od uživatele, a měl by implementovat metody pro jejich provedení.

Takto zvolený návrhový vzor je oblíbený z mnoha důvodů. Umožňuje zorganizovat aplikaci podle typů funkčnosti programu a drží části projektu odlišené. Ale také sebou nese nevýhody:

- je stavový – model je závislý na stavu, takže model může aktualizovat *View* při změně;
- nemá jednoznačnou interpretaci – každý framework používá svoji vlastní variaci;
- kam dát *logging* – není jednoznačné, kde řešit logování dat, která nejsou součástí modelu.

Stavy návrhu jsou hlavní problém, při každé změně se musí projít celý kruh a je třeba dojít k restartování celého obrazu. Více o MVC je možné najít na [3]. Existuje také několik variací modelu. Ty byly změněny pro specifické použití. Následuje výpis těchto vzorů:

- *Model-View-View-Controller* – návrh řeší problém, kde normální *View* řeší jak presentaci, tak presentační datovou strukturu. Je rozdíl mezi informacemi, které zobrazujeme, a informacemi, které připravujeme na zobrazení. Toto řešení je rozdělí do dvou oddělených kategorií.

- *Hierarchical Model-View-Controller* – vzor je podobný, ale je zde možnost seskupit je dohromady. Takže každá strana může mít své vlastní zázemí.
- *Model-View-Adapter* – změna z *Cotroller* na *Adapter* postaví tuto část mezi *Model* a *View.* Tím se oddělí *View* od *Model* a je možné je lépe oddělit.
- *Model-View-Presenter* – nápad zde je takový, že *Presenter* nereaguje s uživatelem. Veškerá komunikace je řešena s částí *View*, které řeší komunikaci s uživatelem a potom ji předává *Pesenter* třetině.

3. Bezpečnost

Webové aplikace je potřeba chránit proti útokům zvenčí. Nejlépe je na tuto skutečnost myslet už při vývoji, ale spousta vývojářů to nedělá a odkládá bezpečnost na poslední místo. V následujících kapitolách se bude rozebírat, jaké útoky mohou nastat a jak aplikace zabezpečit.

3.1 Bezpečnostní rizika

Počítačový experti a podvodníci jsou v neustálém boji o internetová data. Každý rok se na internetu objeví nová nebezpečí. Ta mohou odhalit věci, které jsme odhalit nechtěli. Proti těmto útokům jsou obvykle vyvinuty obrany. Ale neschopnost vývojářů používat poslední možné ochrany při vývoji je největším úskalím pro zabezpečení internetu, a tak frekvence útoků a úspěšných krádeží dat stále roste. Dále budou popsány konkrétní hrozby a způsoby, jak jim nejlépe předejít.

3.1.1 Špatná konfigurace

Tento útok nastává, když vývojář zapomene v systémech původní heslo nebo jiný přihlašovací údaj. Můžeme tomu předejít častým spravováním všech komponent, změnou původní konfigurace nebo častými penetrační testy

3.1.2 Injekční útoky

Webová aplikace náchylná na tento útok má obvykle neošetřené vstupy. Aplikace pracuje se vstupem uživatele bez nutných kroků, které by útoku předešly. Útočníci pak mohou napsat kód do vstupních polí, která aplikace nabízí, a tím se dostat dovnitř. Těmto útokům se dá zabránit tak, že budeme programovat zvlášť vstupní pole a příkazy pro aplikaci nebo databázi. Také můžeme používat zabezpečené API (Application Programming Interface). Další možností je filtrovat a zabezpečit všechna vstupní pole. Bylo řečeno [3], že pokud je systém vystaven útoku zvenčí, je nutné implementovat opatření pro bezpečný provoz v rámci OCPP.

3.1.3 Prolomená autentizace

Dalším oblíbeným způsobem, jak napadnout webovou aplikaci, je její autentizace. Tomu se může předejít ukončením možnosti registrace po uplynutí doby

bez žádné aktivity, zrušením ID registrace jakmile přihlášení proběhlo a uděláním lepších limitů pro složitost hesel.

3.1.4 Cross Site Scripting (XSS)

Jedná se o injekční útok od klientské strany. Podstatou útoku je vložení kódu do webové aplikace, který se projeví po určité době na klientově straně. Každá aplikace, která nekontroluje nedůvěryhodná data, je náchylná tomuto útoku. Chránit se můžeme následovně. Zakódovat veškerá data od zákazníků. Dále můžeme povolit pouze používání písmen a čísel pomocí whitelist. V neposlední řadě je možné používat knihovny, které automaticky čistí data.

3.1.5 Insecure Indirect Object References

Jedná se o manipulaci *URL*. Útočník dostane přístup k datům ostatních zákazníků. Aplikace je tomuto útoku slabá, pokud někdo může měnit *URL* a dostat se k důležitým stránkám bez přihlášení. Vyřešit můžeme předáváním objektů přes *POST*, *GET* namísto *URL*. Také lze zařídit lepší autorizace s častými otázky v průběhu ostatních stránek, případně změnit chybné hlášky tak, aby neobsahovaly údaje o uživatelích.

3.1.6 Chybějící kontrola přístupu

Tato chyba je podobná problému *IDOR*. Na rozdíl od *IDOR* tato chyba dává útočníkům přístup ke speciálním funkcím. Tomu můžeme předejít povolením změny přístupových práv, s tím můžeme práva odebrat nebo přidat, kdy je potřeba. Implementujeme adekvátní přístupové ochrany. Odebereme přístup ke všem přístupovým možnostem a funkcím a necháme je jen administrátorovi.

3.1.7 Špatné přesměrování

Většina stran používá přesměrování. Tuto možnost můžou někteří použít na přesměrování na stranu s viry. Předcházíme následujícími způsoby. Vyhnout se přesměrování pokud je to možné. Dát cíli jiné hodnoty než normální URL.

4. Vývoj aplikace

V následující části práce se bude rozebírat vlastní aplikace, její příprava, řešení a výsledky.

4.1 Návrh řešení

Pro řešení problému byl zvolen návrhový vzor Vodopád. Aplikace nevyžaduje více verzí řešení nebo průběžné presentování výsledků. Model Vodopád je jednoduchým přístupem, který je dostačující.

Dle návrhového vzoru byl postup rozdělen do dílčích částí:

- Početí – v této fázi se probere zadání a požadavky, které budou kladeny na program a jeho tvorbu.
- Zahájení – začátek plánování, požadavky na systém, počítače a další potřebné součástky.
- Analýza – sem patří modely, schémata programu a jeho plánování v částech.
- Design – zde jsou obsaženy struktura programu a jeho kompletní navržení.
- Tvorba – tato část obsahuje vlastní programování aplikace.
- Testování – ladění a ověřování aplikace.
- Nasazení – způsob prezentování a uvedení do provozu.
- Údržba – co je nutné k používání a provozu aplikace.

4.2 Početí

Úkolem práce bylo vytvořit aplikaci, která bude pracovat jako server pro dobíjecí stanici. Program měl dodržovat standard *OCPP* implementován na dobíjecí stanci, konkrétně *OCPP 1.5*.

Základním požadavkem serveru je tedy schopnost komunikovat se stanicí a ovládat ji v provozu. Dále by měl server uchovávat informace o uživatelích a jejich dobíjení. Administrátor bude ovládat server pomocí uživatelského rozhraní.

4.2.1 Specifikace funkcí programu

Podle standardu *OCPP 1.5* musí mít systém následující funkce:

- Dobíjecí stanice
    - *Authorize* – slouží k přístupu uživatele do systému.
    - *Boot Notification* – zpráva o nastartování stanice.
    - *Data Transfer* – funkce pro případné zprávy, které nejsou obsaženy v jiných funkcích.
    - *Diagnostics* – nástroj pro inspekci systému.
    - *Status Notification* – udělení přítomného stavu.
    - *Firmware Status Notification* – informace o softwaru.
    - *Heartbeat* – zjištění, jestli je stanice aktivní.
    - *Meter Values* – předání informace o naměřeném odběru energie po nabití.
    - *Start Transaction* – zahájení transakce, tato funkce začíná proces interakce s uživatelem.
    - *Stop Transaction* – ukončení transakce.
- Centrální systém
    - *Cancel Reservation* – zrušení rezervace.
    - *Change Availability* – změna stavu přístupu.
    - *Change Configuration* – změna nastavení.
    - *Clear Chache* – vyčištění mezipaměti.
    - *Data Transfer* – funkce pro data, která nevyhovují jiným funkcím.
    - *Get Configuration* – výpis nastavení.
    - *Get Diagnostics* – zobrazení stavu.
    - *Get Local List Version* – informace o autorizačním listu
    - *Remote Start Transaction* – začátek komunikace s uživatelem a jeho požadavky ze strany serveru.
    - *Remote Stop Transaction* – konec komunikace o dobíjení.
    - *Reserve Now* – reservace místa.

- *Reset* – reset stanice.
- *Send Local List* – poslání autorizačního listu dobíjecí stanici.
- *Unlock Connector* – příkaz pro povolení dobíjecího slotu.
- *Update Firmware* – vylepšení Firmware.

Specifikace je dále podrobněji popsána ve standardu *OCPP 1.5*. K potřebám této práce je součástí programu *WSDL* soubor, který má všechny funkce podrobně vypsané. Funkce mají své vlastní parametry a také metody pro použití.

Příklad výměny mezi dobíjecí stanicí a serverem může být následující. Dobíjecí stanice pošle *Authorize.req(),* centrální systém odpoví *Authorize.conf()*. Dále transakce pokračuje, *StartTransaction.req()* a *StartTransaction.conf()*. Tímto stanice začne nabíjet auto. Pro ukončení se znovu pošle *Authorize,* následně *StopTransaction*.

Účelem je takto definovat výměny mezi dobíjecí stanicí a serverem pomocí standardu *OCPP*. Funkce musí být popsány se svými názvy a parametry podle protokolu a musí dodržovat správné pořadí.

4.2.2 Požadavky aplikace

Uživatel se bude do aplikace přihlašovat přes uživatelské rozhraní. To by mělo obsahovat přístup do aplikace. Dále ovládání pro funkce serveru. Aplikace musí umět posílat požadavky a přijímat je. Také je musí umět zpracovat.

Pro zpracování dat bude sloužit databáze, do které se budou ukládat data z transakcí. Ta by měla být přístupná z uživatelského rozhraní.

Aplikace by také měla být zabezpečena. Přístup do databáze by neměl být jednouchý. Přihlašování by mělo probíhat pod heslem.

4.3 Zahájení

4.3.1 Volba hardware

Pro používání aplikace bude potřeba počítač. Ten také bude potřeba k jejímu vývoji. K práci byli poskytnuty programy simulující chod serveru. Tyto programy by měly pracovat na jiném počítači než zkoušený server. Celkově je tedy nutné mít dva počítače pro tvorbu a testování.

Připojení k dobíjecí stanici je možné přes LAN. To bylo také zvoleno pro komunikaci mezi počítači.

4.3.2 Volba nástrojů

Práce navazuje na projekt, který byl napsán v *Javě*. Jednalo se o konzolovou aplikaci, kde se mohl zadat příkaz ze specifikace *OCPP* a aplikace vypsala daný příkaz na konzoli. Projekt nebyl přímo použit, protože *Spring Boot* nabízí možnost vygenerovat objekty z *WSDL* souborů. Ale programování zůstalo v jazyce *Java*.

Java byla zvolena, protože je to jazyk používaný nejvíce *OCPA*. Jejich fórum má nejvíce diskuzí v *Javě*. Je to objektově orientovaný jazyk, to je výhodou pro to, jak byl protokol *OCPP* navržen. Jazyk je plně dostačující pro navržení serverové i klientské části. Aplikace je tedy kompletně napsaná v *Javě*.

Pro jednodušší zacházení s knihovnami byl zvolen *Maven* builder, který byl použit ke spravování aplikace a jejího vývoje. To usnadnilo vytvoření projektu a jeho závislostí na knihovnách a pluginech.

Dále byl zvolen framework *Spring*, zejména pro usnadnění práce se zaváděním databáze a webového serveru. Je také použit ke konfiguraci programu. Spravuje veškeré třídy a objekty, které se v projektu nacházejí. *Spring Boot* byl přidán do návrhu pro spouštění aplikace a snadnější zavádění komponent. Pro sestavení projektu byl zvolen *Spring Initialzer*.

Dále byly vybrány následující knihovny:

- Developer Tools

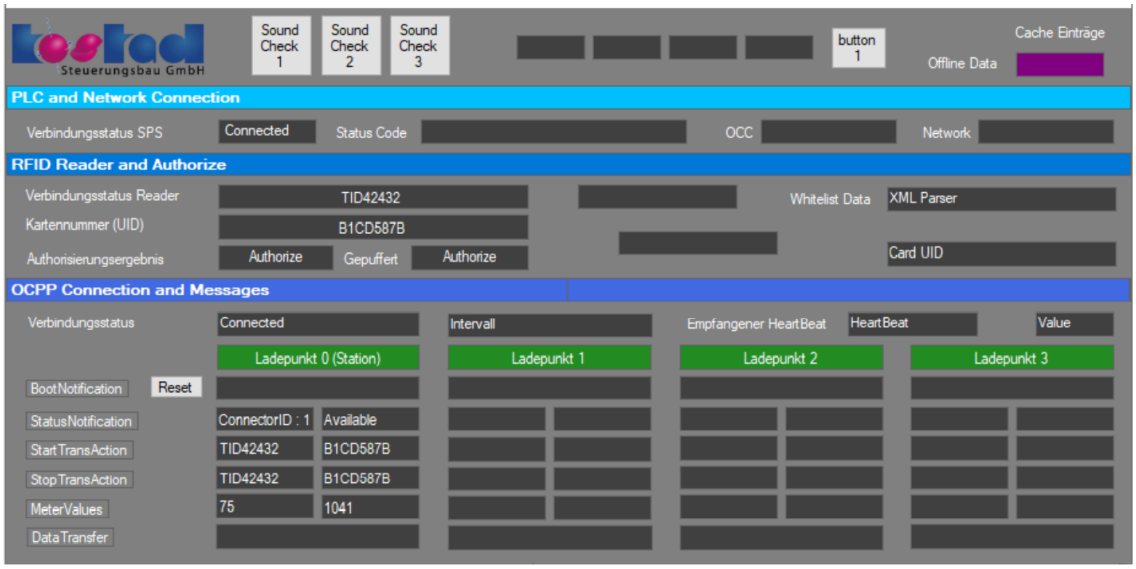
    - Spring Boot DevTools – pomocí DevTools bylo možné odchytit změny programu při vývoji a restartovat aplikaci pro zobrazení výsledku.
    - Lombok – mimo jiné tato knihovna nabízí možnost jednodušších metod typu *getter* a *setter*.
- Web
    - Spring Web – nezbytné pro webové servisy. Volba byla vybrána pro svou jednoduchost.
- SQL
    - Spring Data JPA – použito pro *repository* třídu a pro definování databáze.
    - H2 Database – databáze zvolena pro svou jednoduchost a bezpečnost.
- Input Output
    - Validation – přidáno pro testování výstupů aplikace.

4.3.3 Programovací prostředí

Pro vyhotovení projektu bylo použito vývojové prostředí IntelliJ IDEA. Prostředí bylo zvoleno pro práci s knihovnami projektu a po framework Spring Boot. Nabízí také pohodlné testování, kterého bylo při tvorbě práce využito.

Dále byl použit program Postman, který slouží pro generování dotazů a odpovědí pro práci se serverem. Pomocí tohoto programu se odzkoušelo, zda server reaguje tak, jak má, jestli se do databáze ukládají data a ostatní záležitosti.

Následně bylo využito simulačních programů od firmy Siemens, kde poskytly jednoduchou simulaci klienta i serveru. S těmito programy bylo možné testovat server bez připojení na dobíjející stanici. Pro simulaci byl využit emulátor Hercules a Tibernium Charger.



Obrázek 5: Tiberniu Charger

4.4 Analýza

4.4.1 Návrhový vzor

Pro návrh aplikace byl předně zvolen přístup Model-View-Controller. Aplikace se budou připojovat pouze jedna na jeden server, který vždy komunikuje podle protokolu *OCPP* jen stylem *request* a *confirmation*, z toho důvodu není potřeba používat jiné návrhové vzory.

4.4.1 Databáze

Databáze bude obsahovat 2 tabulky. Jednu pro rezervace a jednu pro data z transakcí. Uživatelé budou mít následující strukturu:

- *Id* – identifikace uživatele.
- *Description* – slouží pro jméno nebo přezdívku či jiný popis uživatele rezervujícího místo.
- *Active* – pro možnost rezervace termínu pro dobíjení.
- *CreatedDate* – datum přidání rezervace.
- *RezervedDate* – datum rezervace.

Tabulka se záznamy bude vytvořena podobným způsobem:

- Id – Identifikace uživatele.
- IdT – Identifikace transakce
- MeterValue – naměřené nabíjení

4.5 Design

4.5.1 Balíčky

Pro tvorbu aplikace byl využit framework Spring, který také má svoji metodiku. Program se jako každá jiná aplikace v *Javě* spouští pomocí *main* třídy v hlavním balíčku. Ale většina logiky byla rozdělena do jiných balíčků:

- *Config* – balíček obsahuje konfigurační soubory pro práci s frameworkem *Spring*. Třídy zde jsou zahrnuty pod anotací *@comonent*, jednotlivé komponenty jsou pak pod anotací *@bean*, nebo *@autowired*
- *Controllers* – zde bude umístěna *Controllers* vrstva návrhového vzoru a pro *Spring Framework.*
- *Models* – toto místo slouží k ukládání modelů. Je pro databáze a pro strukturu dat, která se bude zobrazovat ve složce *resources*.
- *Repositories* – balíček pro kolekce objektů.

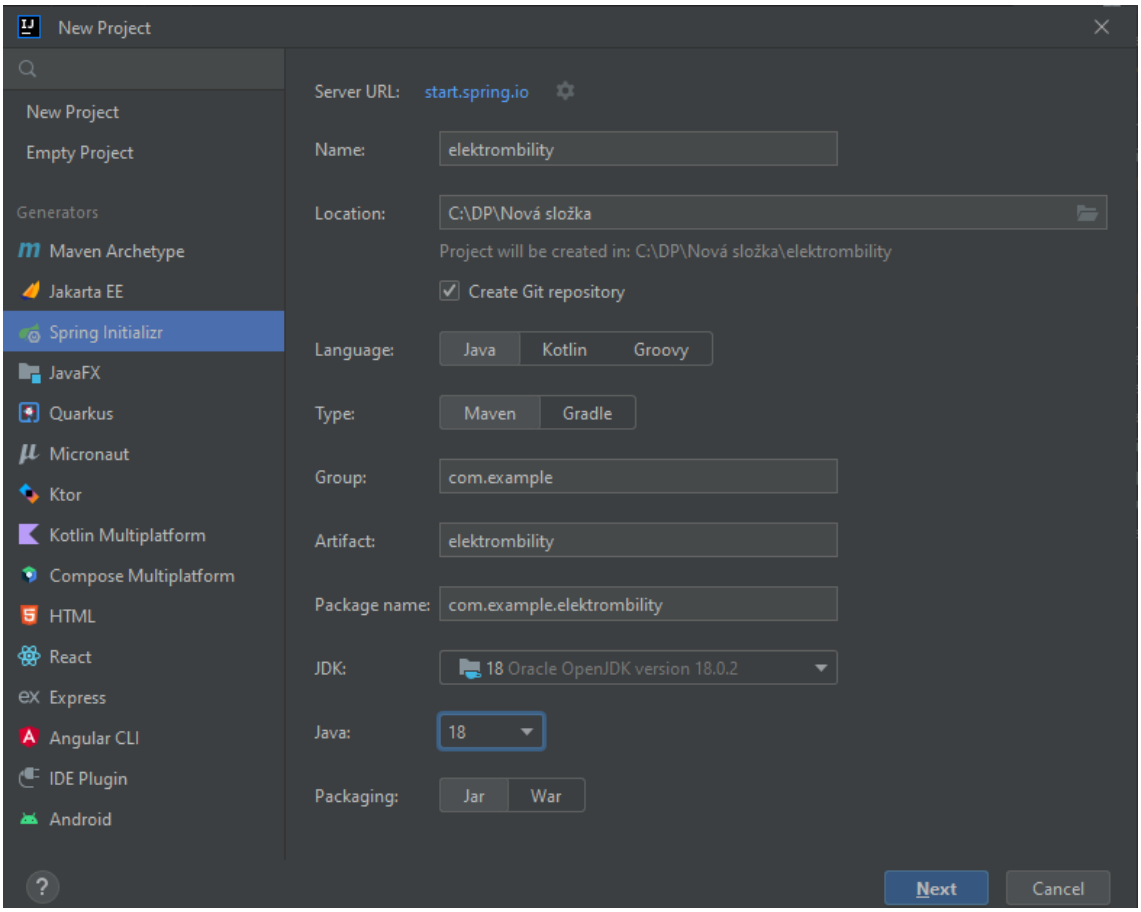
Dále se také bude používat složka *Resources.* V té se bude nacházet *View* část modelu. Také je zde *WSDL* soubor pro práci s *OCPP*.

4.6 Postup řešení

Zde se popisuje postup, který byl použit při tvorbě. Následující kapitoly obsahují pojednání o zprovoznění projektu a jeho zavedení do provozu.

4.6.1 Spring Initializr

Pro prvotní zavedení se použil nástroj *Spring Intializr*. Ten vytvořil projekt podle požadavků pro tvorbu. Byla zde zvolena verze jazyku Java 18. Způsob balíčků byl zvolen stylem Jar. A projekt byl vytvořen jako typ *Maven Project*. Verze *Spring Boot* byla ponechána jako 2.7.3. Následně se doplnily potřebné knihovny a začátek projektu byl hotov.



Obrázek 6: Initializer

Po těchto volbách už nezbylo nic jiného, než vygenerovat projekt. Tímto se vytvořilo prostředí pro následující programování.

4.6.2 Tvorba Databáze

Dalším krokem bylo definování vlastností databáze v balíčku classes. Aplikace pracuje s vestavěnou databází H2. Zde byl zvolen název databáze a další vlastnosti. Byl zde povolen ovladač pro databázi, který je přístupný přes webový prohlížeč. Byl zde zakázán přístup zvenčí. Tím se omezilo riziko napadení databáze, protože databáze pracuje uvnitř aplikace a nemá vlastní server.

```
server.port=8080

# setup local h2 database config
spring.datasource.url=jdbc:h2:file:./data/emDB
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=admin
spring.datasource.password=admin
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# setup local h2 database console
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.h2.console.settings.web-allow-others=false

# setup local h2 tables on startup
spring.jpa.hibernate.ddl-auto=update

# allow devtools restart support
spring.devtools.restart.enabled=true
```

Obrázek 7: nastavení DB

Dále bylo nutné definovat model databáze. Za tímto účelem byl vytvořen nový soubor Java, který obsahuje popis tabulek pro databázi. Ten byl umístěn do nové složky *models*, která obsahuje vrstvu modelujících aplikací. Kromě Spring anotace obsahuje soubor informace o tabulkách v databázi. Také se zde nacházejí metody pro odběr a přidání informací do objektu.

Byly vytvořeny tabulky pro ukládání informací o zákaznících a také jejich transakce s dobíjecí stanicí. Ty se pak mohou z databáze zobrazit nebo může dojít

k úpravě pomocí *CRUD* operací. Pro inicializaci byly vytvořeny počáteční Seed metody, které vytvořily první záznamy.

```
private void loadSeedData() {
    if (userRepository.count() == 0) {
        TabUser user1 = new TabUser( description: "Seed1");
        TabUser user2 = new TabUser( description: "Seed2");

        userRepository.save(user1);
        userRepository.save(user2);
    }

    logger.info("Number of tabUsers: {}", userRepository.count());
}
```

Obrázek 9: Seed

Logika *CRUD* byla umístěna do balíčku *Controllers*. Jsou zde třídy zvlášť pro samotné entity, a jejich skupinové uspořádání. Vše pracuje přes anotace *Spring*. Tvorba modelu z modelu tříd do tabulek proběhla pomocí anotace *@GetMapping* a *@PostMapping*. Pro zasílání zpráv mezi serverem a stanicí zde byla použita třída s anotací *@ResponsePayload*.

```
@PostMapping("/tab")
public String createTabUser(@Valid TabUser tabUser, BindingResult result, Model model) {
    if (result.hasErrors()) {
        return "add-tab-user";
    }

    tabUser.setCreatedDate(Instant.now());
    tabUser.setModifiedDate(Instant.now());
    userRepository.save(tabUser);
    return "redirect:/";
}
```

Obrázek 10: CRUD

4.6.3 Resources

Pro zobrazení informací byly použity *HTML* soubory umístěny ve složce *Resources*, následovně *Templates*. Knihovna *Webjjars* zajišťuje propojení se *Spring Frameworkem*. Díky tomu bylo možné zobrazit data z modelové vrstvy do zobrazení.

```
<script th:src="@{/webjars/jquery/3.6.0/jquery.min.js}"></script>
<script th:src="@{/webjars/bootstrap/3.4.1/js/bootstrap.min.js}"></script>
```

Obrázek 11: Webjjars

Další částí *Resources* je *WSDL* soubor, který byl použit pro vygenerování tříd nezbytných pro vytvoření komunikace. Toho bylo docíleno pomocí pluginu programovacího prostředí, který to udělal automaticky. Tyto třídy pak byly použity v určitých třídách balíčku *Contollers* pro dokončení logiky.

4.7 Testování

K testování byla použita knihovna *JUnit* se *Spring Boot.* Byly vytvořeny testovací metody pro metody vracející hodnoty, aby se zjistilo, jestli hlásí správnou odezvu. Samotné ladění a debugging byly použity ve vývojovém prostředí pomocí značení řádků a jejich krokováním.

Následně byla aplikace vyzkoušena za běhu. První fáze měla za úkol otestovat komunikaci aplikačního serveru a klientu na localhost. Další etapa byla pro vyzkoušení virtuálního systému dodaného firmou Siemens.

4.8 Používání aplikace

4.8.1 Vytvoření spustitelné aplikace

K vytvoření bylo použito vývojové prostředí. Jako výstup byl zvolen program typu *Jar,* který je spustitelný. Po spuštění se spustí aplikace a na localhost v prohlížeči se zpřístupní uživatelské prostředí.

4.8.2 Přihlášení do systému

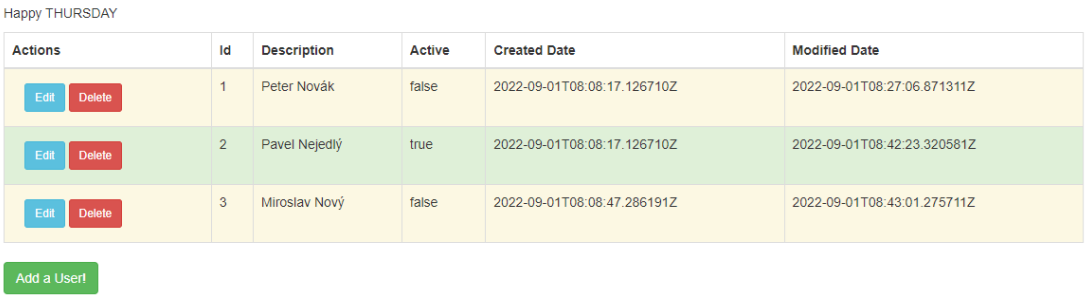
Po vyplnění přihlašovacích údajů se uživatel dostane dále do systému.

4.8.3 Používání systému

V této kapitole se bude rozebírat, jak se aplikace používá a jaké má možnosti. Je rozdělena do 3 skupin.

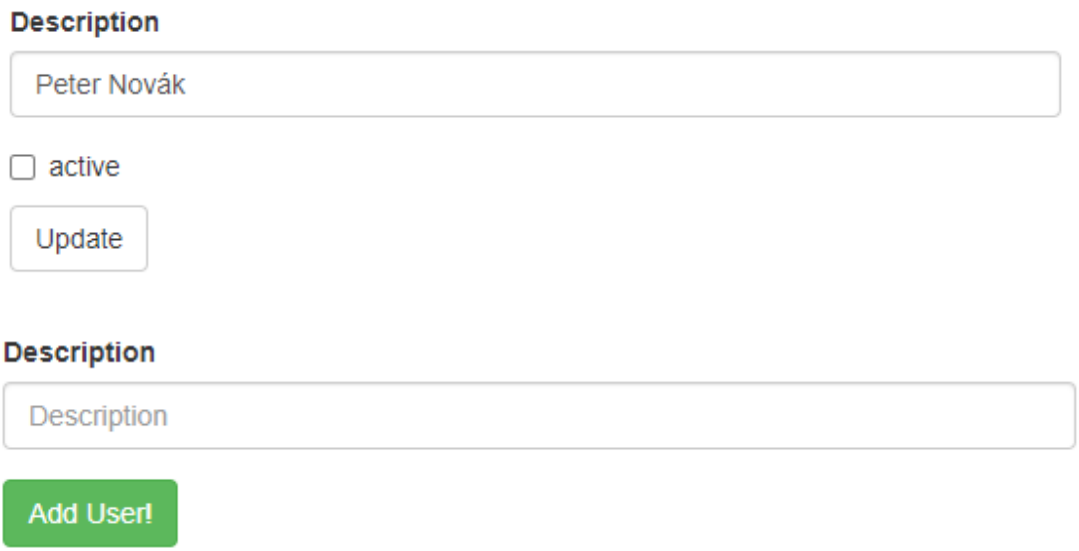
4.8.4 List rezervací

Po přihlášení je jako první vidět list uživatelů. Jedná se o všechny zákazníky, kteří jsou zaregistrováni do systému. Data jsou uchovávána v databázi H2 a mohou být změněna administrátorem pomocí aplikace.



Obrázek 12: List rezervací

Zde je možné přidat nebo ubrat rezervace. Po stisknutí tlačítka se strana přesměruje na přidání nového záznamu. Ten můžeme vyplnit nebo upravit.



4.8.5 Obrázek 13: Přidání reservace

4.8.6 List záznamů dobíjení

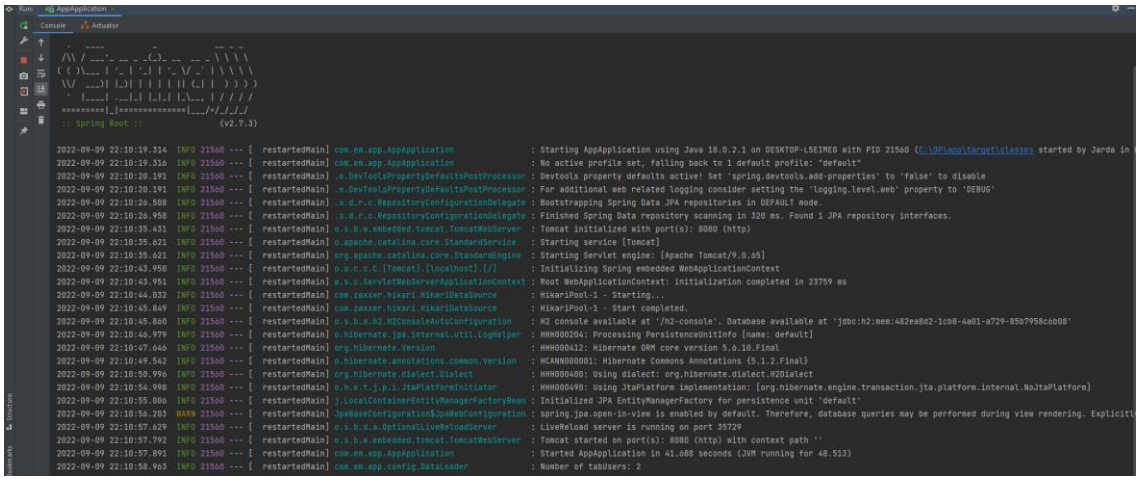
List pracuje na podobném principu. Také se mohou upravovat a měnit záznamy.

5. Závěr

Tato kapitola bude souhrnem všech úspěchů a neúspěchů programu.

5.1 Úspěchy

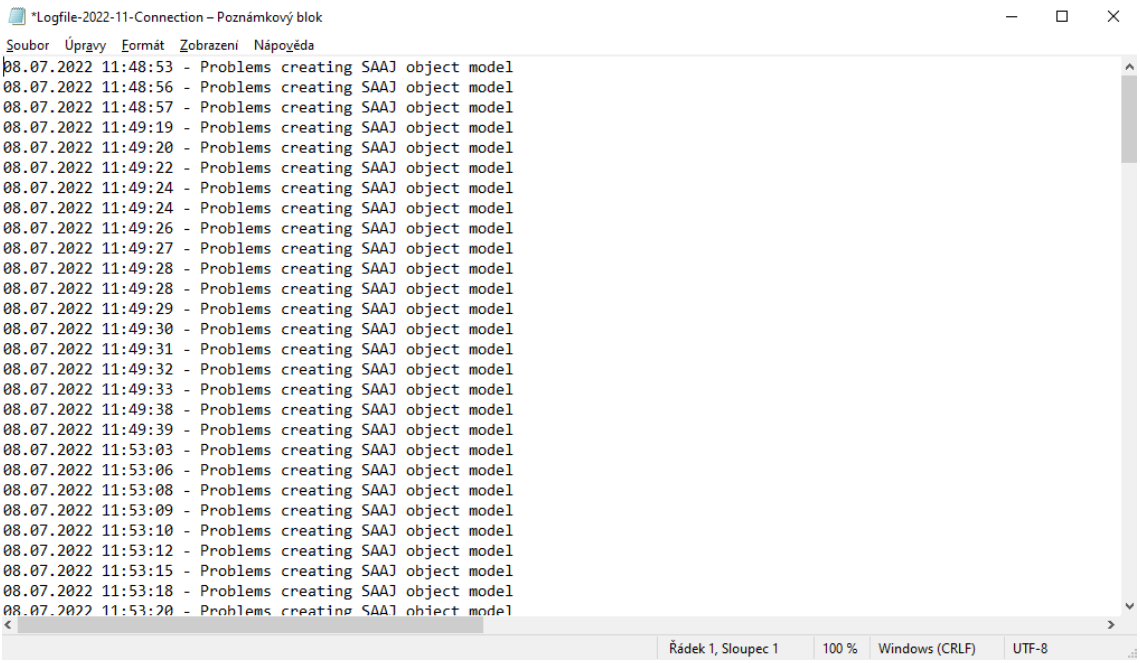
.Podařilo se navrhnout projekt a vytvořit jej za pomoci vývojového prostředí a zvolených nástrojů. Inicializace databáze a prvních dat byla úspěšná. Databáze je dostupná a lze ji ovládat na *localhost* přes zvolenou adresu. *Localhost* pro zobrazení tabulek také pracuje, jak by měl. Data lze zobrazit, mohou se změnit pomocí *CRUD* operací.



Obrázek 14: Úspěšný springboot

5.2 Nedostatky

Propojení s klientem zůstalo na bodu navázání spojení na localhost. Localhost se podařilo propojit, ale funkce napsané tak, jak by měly být, nebylo možné dát na virtuální soubory od Siemens. S největší pravděpodobností jsou stále chybně napsané metody a jejich parametry.



Obrázek 15: Client log

5.3 Pokračování

Pokračováním by bylo zjištění problému připojení a dokončení funkcí pro komunikaci se stanicí.

Seznam použité literatury

[1] Open smart charging protocols. In: *Open Charge Alliance* [online]. Arnhem, Nizozemsko, Businesspark ArnHem Buiten, 2018 [cit. 16.9.2022]. Dostupné z: https://www.openchargealliance.org/protocols/ocpp-15/

[2] Systems development life cycle. In: Wikipedia the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 16.9.2022]. Dostupné z: https://en.wikipedia.org/wiki/Systems_development_life_cycle

[3] MVC aplikace & presentery. In: Nette.org. [Online]. Turkey, 13.3.2012. [cit. 16.9.2022]. Dostupné z: http://doc.nette.org/cs/presenters#toc-sablony.